TensorFlow Test Exam (Examination 20/21)

This is another self-test examination. Notice you are not able to answer questions 13-20, please ignore them.

Beginn: 01.12.2023, 10:10
Ende: 06.12.2024, 16:30

Kurs: Implementing ANNs with TensorFlow (8.3304)
Semester: WiSe 2023/24
Lehrende: Prof. Dr. Elia Bruni, Serwan Jassim

Name: Hendrik Kremer

1. Biological and artificial neurons

1,5 Punkte

Artificial Neural Networks are inspired from biological neural networks. Which mechanisms do both biological and artificial neurons have in common?

Inputs can receive different weightings and can result in enhancing or reducing the output signal.	[✓ True]	False	keine Antwort
In the process of learning, new connections between neurons are formed.	True	[✓ False]	keine Antwort
The information is encoded in the strength of the output.	True	[✓ False]	keine Antwort

Vorsicht: Falsche Antworten geben Punktabzug!

2. Application

10 Punkte

You are a passionate hiker and encounter a beautiful mushroom in the forest. Given the outside appearance of it, its *cap size, cap color, stem length, odor and whether it has gills (Lamellen) or not*, you want to find out whether it is safe to snack this little funghi. Luckily, you have your laptop with you and a large data set with the same categories for many other funghi (excluding this one of course because it's a real rarity!). So you come up with the idea to build a NN model for this funghi dataset and predict whether the mushroom in front of you is *edible* or *poisonous*.
(For simplicity, you can conceive 'color' and 'odor' as categorical variables, with each 10 options.)

1. How do you preprocess this data in order to train a NN on it?
   (shape of input vector, data representation of input and target, transformations on the dataset, ect.)
2. Which type of task is this?
3. Which type of architecture is useful?
4. Which loss function(s) would you use for this task?
5. Which activation and output function do you use?

Shortly give reasons for your choices.

Lösung des Teilnehmers:

Data Preprocessing:

One-Hot Encoding:

- Encode categorical variables like 'color' and 'odor' using one-hot encoding. This transforms each categorical variable into a binary vector, making it suitable for neural networks.

Normalization:

- Normalize numerical features like cap size and stem length to a similar scale, typically between 0 and 1, to ensure that the model is not sensitive to the magnitude of these features.

Handling Missing Data:

- If there is any missing data, decide on an appropriate strategy, such as imputation or removal of instances with missing values.

Creating Input Vector:

- Concatenate all the one-hot encoded vectors and normalized numerical features to form the input vector for the neural network.

Target Encoding:

- Encode the target variable ('edible' or 'poisonous') into numerical values, like 0 and 1, for the model to learn effectively.

Type of Task:

- This is a **binary classification task** since the goal is to predict whether the mushroom is edible (class 0) or poisonous (class 1).

Type of Architecture:

- A **feedforward neural network (FNN)** with multiple hidden layers should be suitable for this task. You can experiment with different architectures based on the size of your dataset.

Loss Function(s):

- For binary classification, the **binary cross-entropy loss** is commonly used. It measures the difference between predicted probabilities and actual class labels.

Activation and Output Function:

- Use **ReLU (Rectified Linear Unit)** as the activation function for hidden layers. It helps introduce non-linearity to the model.
- For the output layer, since it's a binary classification problem, use **sigmoid activation function**. This squashes the output between 0 and 1, representing the probability of the mushroom being poisonous.

Model Training Considerations:

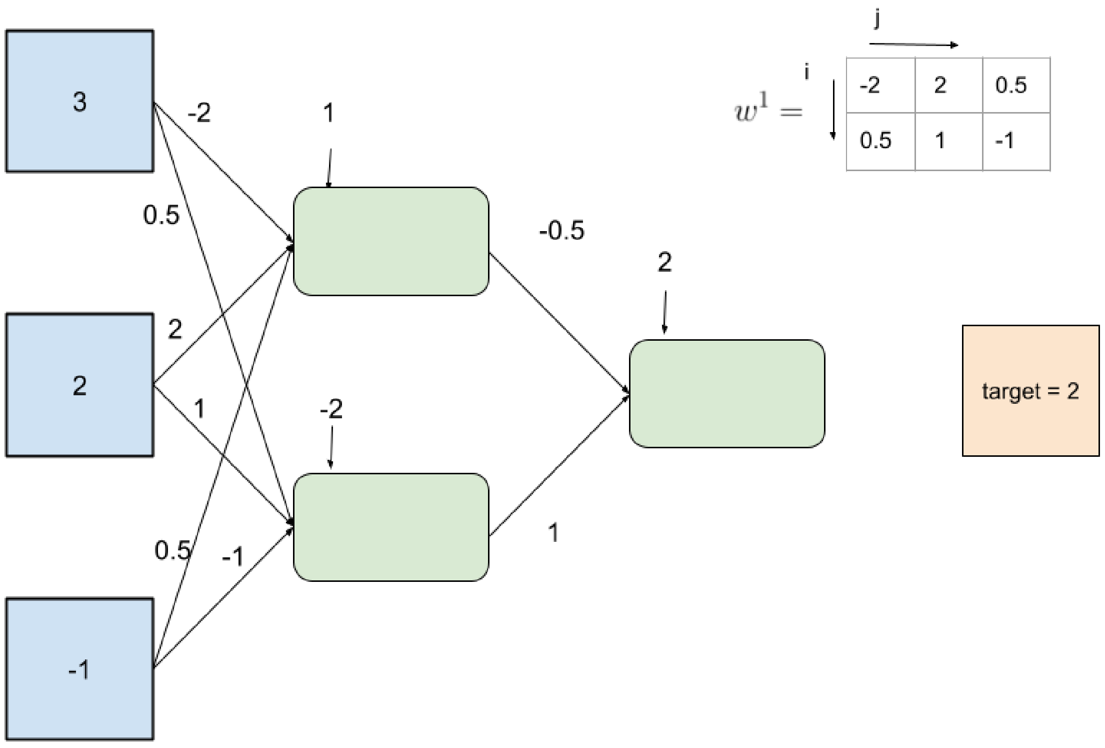
- Split your dataset into training and validation sets to assess the model's performance on unseen data.
- Monitor the training process for potential overfitting and consider techniques such as dropout or regularization to address it.
- Adjust hyperparameters (learning rate, number of layers, neurons per layer) based on the model's performance on the validation set.

3. Backpropagation

10 Punkte

Compute the forward step for the network below. Note the drives and activations for each of the neurons.
Then compute a backward step using Backpropagation, noting the deltas and the updated weights.
Learning rate = 0.1, **Activation** = ReLU, **Loss** = $\frac{1}{2}MSE$

Notation:
- Superscripts index the layers.
- Subscripts index the weight matrices and vectors like shown exemplarily in the table in the upper left corner.



$$w^1 = \begin{array}{|c|c|c|} \hline -2 & 2 & 0.5 \\ \hline 0.5 & 1 & -1 \\ \hline \end{array}$$

Forward Step:

$d_1^1 = \underline{\;-1.5\;}$
$a_1^1 = \underline{\;0\;}$

$d_2^1 = \underline{\;2.5\;}$
$a_2^1 = \underline{\;2.5\;}$

$d_1^2 = \underline{\;4.5\;}$
$a_1^2 = \underline{\;4.5\;}$

Backward Step:

$\delta_1^1 = \underline{\;0\;}$
$\delta_2^1 = \underline{\;2.5\;}$
$\delta_1^2 = \underline{\;2.5\;}$

Updated Weights:

$w_{1,1}^2 = \underline{\;-0.5\;}$
$w_{1,2}^2 = \underline{\;0.375\;}$

$b^2_1 =$ 1.75

$w^1_{1,1} =$ -2
$w^1_{1,2} =$ 2
$w^1_{1,3} =$ 0,5
$b^1_1 =$ 1

$w^1_{2,1} =$ -0,25
$w^1_{2,2} =$ 0,5
$w^1_{2,3} =$ -0,75
$b^1_2 =$ -2,25

4. Optimizers

5 Punkte

Indicate each of the statements below as true or false. False answers will lead to a deduction of points.

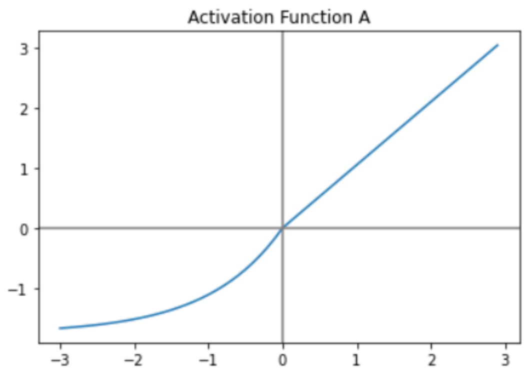
Adding momentum to stochastic gradient descent decreases the chance of getting stuck in local minima.	[✓ True]	False	keine Antwort
Batch learning is computationally more efficient in terms of memory than single-sample learning, because we only have to compute the gradients every batchsize steps	True	[✓ False]	keine Antwort
Batch learning is computationally less efficient in terms of memory than single-sample learning, because we have to compute the gradients for all batchsize items.	[✓ True]	False	keine Antwort
The only difference between Adagrad and SGD with momentum is that Adagrad takes into account all past gradients whereas SGD with momentum only considers the last gradient when computing the new gradient.	True	[✓ False]	keine Antwort
One of Adagrad's shortcomings is that at some point the learning rate is too small and no more learning is possible. Adadelta and RMSProp attempt to solve this problem.	[✓ True]	False	keine Antwort

Vorsicht: Falsche Antworten geben Punktabzug!

5. Activations and Gradients

2,5 Punkte

Select the correct statements about activation functions and gradients.



Linear activation functions are useful in ANNs because their derivative is easy to compute.	○ True	[✓ False]	○ keine Antwort
ReLU is an activation function that is prone to exploding gradients. Leaky ReLU is an adaptation of ReLU to avoid this.	○ True	[✓ False]	○ keine Antwort
Exploding gradients are a common problem for single-saturating functions.	[✓ True]	○ False	○ keine Antwort
The activation function in the graphic above will not cause dying ReLU.	[✓ True]	○ False	○ keine Antwort
The activation function in the graphic above will not cause vanishing gradients.	[✓ True]	○ False	○ keine Antwort

Vorsicht: Falsche Antworten geben Punktabzug!

6. Drives

3 Punkte

Explain why it is useful to center the drives around zero. Describe two techniques to achieve zero-centered drives. (3-6 sentences)

Lösung des Teilnehmers:

Two techniques to achieve zero-centered drives are:

DC Bias Removal: This involves removing any DC (direct current) component from the signal. DC bias can shift the entire signal away from zero. Techniques such as high-pass filtering or subtracting the mean value of the signal can be employed to eliminate the DC bias.

Normalization: Normalizing the signal involves scaling it so that its mean becomes zero. This can be achieved by subtracting the mean value of the signal from each data point and dividing by the standard deviation. Normalization ensures that the signal has a zero mean and a standard deviation of 1, making it centered around zero.

By centering the drives around zero, engineers can simplify the analysis and design of control systems, reduce the risk of saturation in electronic components, and enhance the overall performance of the system.

7. Overfitting

5 Punkte

What is overfitting?

Explain two regularization methods used in deep neural networks that combat overfitting.

Lösung des Teilnehmers:

Overfitting:
Overfitting occurs in machine learning, including deep neural networks, when a model learns the training data too well to the extent that it starts capturing noise or random fluctuations in the data rather than the underlying patterns. As a result, an overfitted model performs well on the training data but poorly on new, unseen data because it has essentially memorized the training set instead of generalizing from it.

Regularization Methods to Combat Overfitting:

L1 and L2 Regularization:
L1 Regularization (Lasso): This technique adds a penalty term to the loss function that is proportional to the absolute values of the model parameters. The regularization term is the sum of the absolute values of the weights. This encourages sparsity in the model by pushing some of the weights to exactly zero, effectively excluding certain features from the model. L1 regularization helps prevent overfitting by simplifying the model and selecting only the most important features.

L2 Regularization (Ridge): L2 regularization adds a penalty term to the loss function that is proportional to the square of the magnitudes of the model parameters. The regularization term is the sum of the squared weights. L2 regularization penalizes large weights but doesn't force them to be exactly zero. It helps prevent overfitting by discouraging overly complex models and limiting the influence of any single feature

8. Convolution

3 Punkte

Calculate the feature map from the image below and the convolutional kernel.
No padding, stride size = 2
(Write the solution in the Text Editor or upload a picture)

Image

```
[[ 0  1  0  0  0]
 [ 4  0  0 -3  0]
 [-5  0 -5  2  0]
 [ 0  0  0  0  0]
 [-5  0  3  0  0]]
```

Kernel

```
[[-2  0  3]
 [ 0  0 -2]
 [-1  0  0]]
```

Lösung des Teilnehmers:

0*-2+0*1+0*0+4*0+0*0+0*-2+-1*-5+0*0+-5*0 = 5
0*-2+0*0+3*0+0*0+-3*0+0*0+-1*-5+0*2+0*0 = 5
-5*-2+0*0+-5*3+0*0+0*+-2*0+-1*-5+0*0+0*3 = 0
-5*-2+0*5+3*0+0*0+0*0+0*-2+-1*3+0*0+*0* = 7

[5,5]
[0,7]

9. CNN

4 Punkte

Select the correct statements about (advanced) convolutional neural networks.

Bottlenecks help avoiding vanishing gradients.	True	[✓ False]	keine Antwort
The number of filters in Dense CNNs is defined by the growth rate.	[✓ True]	False	keine Antwort
Applying Transposed Convolution to a feature map returns the original input image.	True	[✓ False]	keine Antwort
Using pooling layers instead of flattening the feature maps will keep the number of trainable parameters low.	[✓ True]	False	keine Antwort

Vorsicht: Falsche Antworten geben Punktabzug!

10. Advanced Models

10 Punkte

Deeper networks often prove to be better at solving the task at hand. However, increasing depth might also cause problems:

1. Name a common problem that arises in deeper neural networks. Shortly state why this problem is more likely to occur in deeper models. 1P

2. Name two adaptations of the classical CNN and explain one of them in detail, outlining how the architecture is designed to avoid the problem. 4P

3. Why does the problem also arise in simple RNNs that only consist of one layer? 1P

4. Name two adapations of the vanilla RNN and explain one of them in detail, outlining how the architecture is designed to avoid the problem. 4P

Lösung des Teilnehmers:

Common Problem in Deeper Neural Networks:

Vanishing Gradient Problem (1P): The vanishing gradient problem occurs when gradients become extremely small during backpropagation, leading to negligible weight updates in deeper layers. This is more likely to happen in deeper models due to the repeated multiplication of small gradients as they propagate through many layers during training.

Adaptations of Classical CNN:

Dilated (Atrous) Convolution (4P):

Dilated convolution is an adaptation of classical convolutional neural networks (CNNs). In dilated convolution, the kernel has gaps between its values, allowing it to have a larger receptive field without increasing the number of parameters. This helps capture more global information while avoiding the computational cost associated with a significant increase in parameters.

Architecture Design:

- Dilated convolutions introduce a dilation rate parameter that determines the spacing between kernel values. A dilation rate of 1 corresponds to the standard convolution, while larger values increase the gap between values. The output size of the dilated convolution layer is larger than that of the standard convolution, capturing a broader context.

Residual Networks (ResNets) (4P):

ResNets address the vanishing gradient problem by introducing skip connections (or shortcuts) that bypass one or more layers. This enables the gradient to flow directly through the shortcut during backpropagation, mitigating the vanishing gradient problem and allowing for the training of very deep networks.

Architecture Design:

- ResNets use residual blocks, where the input to a block is added to the output before passing through the activation function. This way, even if the gradients through the layers are small, the identity mapping provided by the shortcut allows the information to flow easily through the network.

Problem in Simple RNNs:

Vanishing Gradient Problem in Simple RNNs (1P): The vanishing gradient problem also arises in simple recurrent neural networks (RNNs) that consist of a single layer. This happens because during backpropagation, gradients can diminish as they are multiplied across the recurrent connections over time steps.

Adaptations of Vanilla RNN:

1. **Long Short-Term Memory (LSTM) Networks (4P):**

LSTMs are an adaptation of the vanilla RNN designed to address the vanishing gradient problem and capture long-term dependencies in sequences.

Architecture Design:

- LSTMs include memory cells with gating mechanisms, such as input gates, forget gates, and output gates. These gates control the flow of information into and out of the memory cell, allowing the network to selectively store or discard information over time. The design of LSTMs enables them to capture long-range dependencies and gradients can flow more easily through these memory cells.

These adaptations in both CNNs and RNNs demonstrate how architectural modifications are crucial for overcoming the challenges associated with deep neural networks, such as the vanishing gradient problem.

11. Which architecture?

5 Punkte

Look at the following code.

1. Which architecture is this?
2. Which code snippet(s) made you realize it?
3. Name one application of the architecture.
4. Explain the function of the highlighted class in the architecture. (class ApplePen)

```
### Explain the function of this class in the architecture.
class ApplePen(tf.keras.layers.Layer):
  def __init__(self, AppleDiameter):
    super(ApplePen, self).__init__()
    self.layers_list = [tf.keras.layers.Conv2D(AppleDiameter, kernel_size=1, padding='same'),
                        tf.keras.layers.BatchNormalization(),
                        tf.keras.layers.Activation('relu'),
                        tf.keras.layers.AveragePooling2D(strides=2),
                        tf.keras.layers.Dropout(0.2),
    ]

  def call(self, x, training):
    for layer in self.layers_list:
      x = layer(x, training=training)
    return x
### end of class ###


class Pen(tf.keras.layers.Layer):
  def __init__(self, num_pineapples):
    super(Pen, self).__init__()
    self.layers_list = [tf.keras.layers.BatchNormalization(),
                        tf.keras.layers.Activation('relu'),
                        tf.keras.layers.Conv2D(num_pineapples*4, kernel_size=1, padding='same'),
                        tf.keras.layers.Dropout(0.2),
                        tf.keras.layers.BatchNormalization(),
                        tf.keras.layers.Activation('relu'),
                        tf.keras.layers.Conv2D(num_pineapples, kernel_size=3, padding='same'),
                        tf.keras.layers.Dropout(0.2),
    ]
    self.concat = tf.keras.layers.Concatenate()

  def call(self, x, training):
    y = x
    for layer in self.layers_list:
      x = layer(x, training=training)
    Pen = self.concat([x,y])
    return Pen


class PenPineapple(tf.keras.layers.Layer):
  def __init__(self, num_pen, num_pineapple):
    super(PenPineapple, self).__init__()
    self.layers_list = [Pen(num_pineapple) for _ in range(num_pen)]

  def call(self, x, training):
    y = x
    for block in self.layers_list:
      y = block(y, training=training)
    return y

class PenPineappleApplePen(tf.keras.Model):
  def __init__(self, AppleDiameter=12, num_pineapple=12, num_pens=[12,12,12]):
    super(PenPineappleApplePen, self).__init__()

    self.pre_pen = [tf.keras.layers.Conv2D(AppleDiameter, kernel_size=3, strides=1, padding='same'),
                tf.keras.layers.BatchNormalization(),
                tf.keras.layers.Activation('relu')
    ]

    self.PenPineappleApplePen = []

    for i, num_pen in enumerate(num_pens):

      self.PenPineappleApplePen.append(PenPineapple(num_pen, num_pineapple))
      AppleDiameter += num_pen * num_pineapple


      if i != len(num_pens)-1:
        AppleDiameter // 2
        self.PenPineappleApplePen.append(ApplePen(AppleDiameter))

    self.post_apple =  [tf.keras.layers.GlobalAveragePooling2D(),
                  tf.keras.layers.Dense(units=10, activation='softmax')
    ]


  def call(self, x, training):

    for layer in self.pre_pen:
      x = layer(x, training=training)

    for layer in self.PenPineappleApplePen:
      x = layer(x, training=training)

    for layer in self.post_apple:
      x = layer(x, training=training)

    return x
```

Lösung des Teilnehmers:

Architecture Identification:
This code represents a variation of the Inception (GoogLeNet) architecture, particularly with the inclusion of the custom layers named ApplePen and Pen.

Code Snippets Identifying the Architecture:
Inception Blocks:

The presence of multiple parallel branches with different convolutions and pooling operations, as seen in the Pen and ApplePen classes, resembles the characteristic structure of Inception blocks.
Concatenation Operation:

The use of tf.keras.layers.Concatenate() in the Pen class indicates the concatenation of feature maps from different branches, a key element of Inception architecture.
Application of the Architecture:
Image Classification:

Inception architectures, including variations like this one, are often used for image classification tasks. The ability to capture information at different scales through parallel convolutional branches makes Inception networks effective in recognizing patterns and features in images.
Function of the Highlighted Class (ApplePen):
The ApplePen class serves as a custom layer within the architecture. Its primary function is to define a specific block of operations applied to the input tensor. The operations within the ApplePen class include:

Convolution:

A 1x1 convolution with the specified number of filters (AppleDiameter), which can control the number of output channels.
Batch Normalization:

Batch normalization layer to normalize the activations, stabilizing and accelerating the training process.
ReLU Activation:

Rectified Linear Unit (ReLU) activation function to introduce non-linearity.
Average Pooling:

Average pooling operation with a stride of 2, which downsamples the spatial dimensions of the tensor.
Dropout:

Dropout layer with a dropout rate of 0.2, helping prevent overfitting during training.
The ApplePen class can be considered a building block within the architecture, contributing to the overall feature extraction and hierarchical representation learning. It allows for the incorporation of domain-specific operations, enhancing the expressive power of the neural network.

12. Parameter

4 Punkte

Consider this network:

- Input size (15,10) of shape (timesteps, features)
- Dense Layer with 5 units
- Sigmoid Activation
- LSTM Cell with 15 hidden units
- Dense layer with 10 units
- Softmax Activation

Compute number of trainable parameters of this network. (Write down the computations for each layer, not just the result.)

Write the answer in the text editor or upload a picture of your solution.

Lösung des Teilnehmers:

Input Layer:

- The input layer does not have trainable parameters. It simply passes the input to the next layer.

Dense Layer with 5 units (Sigmoid Activation):

- Parameters for weights: 15 (input features) * 5 (units) = 75
- Parameters for biases: 5
- Total parameters: 75 (weights) + 5 (biases) = 80

LSTM Cell with 15 hidden units:

- Parameters for input weights: 10 (input features) * 15 (hidden units) = 150
- Parameters for recurrent weights: 15 (hidden units) * 15 (hidden units) = 225
- Parameters for biases: 15 (hidden units)
- Total parameters: 150 (input weights) + 225 (recurrent weights) + 15 (biases) = 390

Dense Layer with 10 units (Softmax Activation):

- Parameters for weights: 15 (hidden units from LSTM) * 10 (units) = 150
- Parameters for biases: 10
- Total parameters: 150 (weights) + 10 (biases) = 160

Total Number of Trainable Parameters:

- Summing up the parameters from all layers: 0 (input) + 80 (dense layer) + 390 (LSTM cell) + 160 (dense layer) = 630

13. Variational Autoencoders

3 Punkte

The decoder of Variational Autoencoders is trained to output the parameters of a multidimensional Gaussian distribution.	True	[✓ False]	keine Antwort
Given a specific latent space encoding, the decoder of the VAE will always output the same sample.	[✓ True]	False	keine Antwort
The Kullback-Leibler Divergence is used to calculate the distance between the latent probability distribution and standard normal Gaussian.	[✓ True]	False	keine Antwort
During reparameterization, we introduce an epsilon to the model which expresses the error rate of the VAE model.	True	[✓ False]	keine Antwort
Changing the parameters of the latent distribution to standard normal Gaussian (mean = 0, logvar= 0) is also referred to as “reparameterization trick”.	True	[✓ False]	keine Antwort
Learning a densely packed latent space representation allows the VAE to generate meaningful samples.	[✓ True]	False	keine Antwort

Vorsicht: Falsche Antworten geben Punktabzug!

14. What the loss?

5 Punkte

Shortly explain the GAN loss and the autoencoder loss (1-3 sentences each). Explain the advantages of the GAN loss regarding image generation. (2-5 sentences)

Lösung des Teilnehmers:

GAN Loss (Generative Adversarial Network Loss): GAN loss is the objective function used in Generative Adversarial Networks. It consists of two components: the generator loss and the discriminator loss. The generator aims to produce data that is indistinguishable from real data, while the discriminator tries to differentiate between real and generated data. The objective is for the generator to improve by minimizing the discriminator's ability to distinguish between real and generated samples.

Autoencoder Loss: Autoencoder loss is used in autoencoder models, which consist of an encoder and a decoder. The loss measures the difference between the input and the reconstructed output, encouraging the model to learn a compressed representation of the input data.

Advantages of GAN Loss in Image Generation: GANs excel in generating realistic images due to their adversarial training. The adversarial nature encourages the generator to generate highly realistic samples that can deceive the discriminator. GANs often produce sharper and more visually appealing images compared to autoencoders, making them particularly effective for tasks like image synthesis and generation. The adversarial training dynamic allows GANs to capture intricate details and nuances in the generated content, leading to impressive image quality. However, GANs also come with challenges such as mode collapse and training instability that need to be addressed for optimal performance.

15. GAN

2 Punkte

Select the correct statements about Generative Adversarial Networks.

Mode collapse is indicated by a high generator loss.	○ True	[✓ False]	○ keine Antwort
If discriminator and generator loss converge to zero the GAN has trained successfully.	○ True	[✓ False]	○ keine Antwort
Vanishing gradients in the generator can be caused by an overperforming discriminator.	[✓ True]	○ False	○ keine Antwort
Weight clipping is used in Wasserstein GANs to enforce Lipschitz continuity.	[✓ True]	○ False	○ keine Antwort

Vorsicht: Falsche Antworten geben Punktabzug!

16. Attention

4 Punkte

Explain how Bahdanau attention works in 2-3 sentences. Also mention how it mitigates a common problem of classical sequence2sequence learning.

Lösung des Teilnehmers:

Bahdanau Attention: Bahdanau attention is an attention mechanism introduced to sequence-to-sequence models, particularly in the context of neural machine translation. It dynamically selects relevant parts of the input sequence for each step of decoding. It does so by assigning attention weights to different positions in the input sequence, allowing the model to focus on specific parts during the generation of each element in the output sequence.

Mitigation of Common Problem: Bahdanau attention mitigates the problem of information bottleneck in classical sequence-to-sequence learning. In traditional approaches, a fixed-size context vector is used, which may not capture all the relevant information for decoding. Bahdanau attention addresses this by allowing the model to attend to different parts of the input sequence at each decoding step, effectively overcoming the limitation of fixed context vectors and improving the model's ability to handle variable-length input sequences. This dynamic attention mechanism enhances the model's capacity to capture long-range dependencies and improves overall sequence generation performance.

17. NLP

4 Punkte

What is the main problem in terms of computational efficiency of the training process in CBOW and SkipGram? Explain how negative sampling reduces this problem.

Lösung des Teilnehmers:

Main Problem in Computational Efficiency for CBOW and SkipGram: The main problem in terms of computational efficiency for both Continuous Bag of Words (CBOW) and Skip-Gram models is the large number of softmax computations required during training. The softmax function computes probabilities for all possible words in the vocabulary, which can be computationally expensive and slow, especially for large vocabularies.

Negative Sampling to Improve Efficiency: Negative sampling is a technique used to address the computational inefficiency in training word embedding models like CBOW and Skip-Gram. Instead of computing probabilities for all words in the vocabulary, negative sampling involves sampling a small subset of "negative" words (words not in the context or target) for each training instance. The objective then becomes distinguishing the positive word from the randomly sampled negative words.

How Negative Sampling Works:

Training Objective Adjustment:

- Instead of maximizing the probability of the correct word, the model is trained to distinguish the correct word from randomly sampled negative words.

Reduction in Computational Cost:

- By sampling only a small number of negative words (typically 5-20), as opposed to the entire vocabulary, the computational cost is significantly reduced.

Efficient Word Embeddings:

- Negative sampling allows the model to efficiently learn high-quality word embeddings by focusing on the relevant context and reducing the complexity of the softmax computation.

Approximation of True Objective:

- While negative sampling introduces an approximation to the true objective, the computational gains often outweigh the loss in precision, making it a practical and efficient strategy for training large-scale word embeddings.

Negative sampling is a key optimization technique that enables the training of word embeddings on large datasets, making it feasible to learn high-quality representations of words in a computationally efficient manner.

18. Reverse ML Engineering

12 Punkte

Programming Task: Consider the following sizes. Write a keras model that reproduces those. Make sure that your model is fully functional, i.e. inherits from the right super class and has the necessary functions.

- Input: (64,64,3)
- After layer 1: (32, 32, 16)
- After layer 2: (30, 30, 32)
- After layer 3: (15, 15, 32)
- After layer 4: (7200)
- After layer 5: (10)
- After layer 6: (8192)
- After layer 7: (16, 16, 32)
- After layer 8: (32, 32, 32)
- After layer 9: (64, 64, 64)
- After layer 10: (64, 64, 3)

Copy your code in the text editor below or upload a file. Do not send links to Colab ect.!

Lösung des Teilnehmers:

```
import tensorflow as tf
from tensorflow.keras import layers, models

class CustomModel(tf.keras.Model):
  def __init__(self):
      super(CustomModel, self).__init__()

      # Layer 1
      self.conv1 = layers.Conv2D(16, (3, 3), strides=(2, 2), padding='valid', activation='relu')

      # Layer 2
      self.conv2 = layers.Conv2D(32, (3, 3), padding='valid', activation='relu')

      # Layer 3
      self.pooling1 = layers.MaxPooling2D((2, 2))

      # Layer 4
      self.flatten = layers.Flatten()

      # Layer 5
      self.dense1 = layers.Dense(10, activation='relu')

      # Layer 6
      self.dense2 = layers.Dense(8192, activation='relu')

      # Layer 7
      self.reshape1 = layers.Reshape((16, 16, 32))

      # Layer 8
      self.conv3 = layers.Conv2D(32, (3, 3), padding='same', activation='relu')

      # Layer 9
      self.conv4 = layers.Conv2D(64, (3, 3), padding='same', activation='relu')
```

```
        # Layer 10
        self.conv5 = layers.Conv2D(3, (3, 3), padding='same', activation='sigmoid')

    def call(self, inputs, training=None, mask=None):
        x = self.conv1(inputs)
        x = self.conv2(x)
        x = self.pooling1(x)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.dense2(x)
        x = self.reshape1(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        return x

# Create an instance of the model
model = CustomModel()

# Display model summary
model.build(input_shape=(None, 64, 64, 3))
model.summary()
```

19. Reinforcement Learning

6 Punkte

Check out the two Lunar Lander Reinforcement Learning Environments: https://gym.openai.com/envs/LunarLander-v2/ and https://gym.openai.com/envs/LunarLanderContinuous-v2/ .

1. For https://gym.openai.com/envs/LunarLander-v2/ describe the underlying Markov Decision Process (MDP).

2. What is the difference between the two environments in terms of the MDP?

3. Choosing between Policy Gradients and DQN - which one would you use for https://gym.openai.com/envs/LunarLander-v2/ and https://gym.openai.com/envs/LunarLanderContinuous-v2/ ? Give reasons and explain why you might choose different algorithms between the two environments.

20. Bonus Task

0 Punkte

This is a bonus task - its respective points are not needed for achieving 100% in this exam, but you can achieve up to 2 bonus points. Only work on this task if you have finished all the other tasks. You will need to spend ~30-60 minutes and can not achieve any more than 2 points! But if you are finished with everything, feel free to try your hands on some actual scientific paper and earn those 2 points ;)

Check out the paper "Learning Sparse Neural Networks through L0 Regularization" here:
https://arxiv.org/abs/1712.01312

Summarize the core idea and how the paper achieves it in no more than 10 sentences.